IN THE CLAIMS

Please amend the claims as follows:

1. (original) A method comprising parallelizing execution of loop computations that involve indirectly accessed sparse arrays/matrices using software-pipelining to increase instruction-level parallelism and decrease initiation interval.

2. (currently amended) The method of claim 1, wherein software-pipelining comprises: parallelizing dependent ~~dependence~~ loop body instructions between ~~the~~ subsequent independent iterations.

3. (currently amended) The method of claim 2, wherein decreasing initiation interval comprises:

increasing resource initiation interval ~~and recurrence initiation interval~~, wherein resource initiation interval is based on resource usage of the dependence loop and available processor resources and recurrence initiation interval is based on the instructions in the dependence loop body and latencies of the processor.

4. (currently amended) The method of claim 2 [[1]], wherein the dependent loop body instructions are based on clock cycles.

5. (original) A method comprising:

parallelizing execution of loop computations, by transforming dependence loop instructions into multiple independent iterations, in sparse arrays/matrices using software-pipelining to break recurrence initiation interval such that recurrence initiation interval is substantially closer to resource initiation interval.

6. (original) The method of claim 5, wherein using the software-pipelining to break the recurrence initiation interval comprises implementing loop-body instructions in parallel in the multiple independent iterations.

7. (original) The method of claim 6, wherein the loop-body instructions comprise loop-body cycles.

8. (original) The method of claim 7, wherein the resource initiation interval is based on resource usage of the loop and available processor resources and the recurrence initiation interval is based on the cycles in the dependence loop and latencies of the processor.

9. (currently amended) A method comprising:

performing a run time dependency check during a current iteration using prior computed values obtained from a predetermined number of previous adjacent iterations in a sparse array matrix; and

parallelizing dependent ~~dependence~~ loop instructions between the current iteration and subsequent multiple iterations by using the computed values to make the dependence loop instructions into independent loop iterations so as to increase instruction-level parallelism and reduce recurrence initiation interval in the current iteration as a function of the run time dependency check during [[in]] the current iteration.

10. (original) The method of claim 9, wherein the prior computed values are based on virtual unrolling using a virtual unrolling factor.

11. (currently amended) The method of claim 10, wherein the virtual unrolling factor includes [[is]] three previous iterations.

12. (original) The method of claim 10, wherein the virtual unrolling factor is computed using the recurrence initiation interval and latency in number of cycles of a floating point multiply add operation in the sparse array matrix.

13. (original) The method of claim 9, further comprising:

assigning the prior computed values to a predetermined number of adjacent registers.

14. (original) The method of claim 12, further comprising:

performing register rotation to include computed values obtained from the current iteration; and

repeating performing the run time dependency check and parallelizing the dependence loop instructions to increase the instruction-level parallelism and reduce the recurrence initiation interval in a next iteration.

15. (original) A method comprising:

transforming sparse array matrix code to perform a run time dependency check using a predetermined number of prior computed values;

software-pipelining the transformed sparse array matrix code to perform the run time dependency check in a current iteration using the predetermined number of prior computed values; and

software-pipelining to parallelize multiple iterations by overlapping execution of dependence loop instructions in the prior computed values to reduce recurrence initiation interval in the sparse array matrix based on the run time dependency check.

16. (currently amended) The method of claim 15, wherein software-pipelining the transformed sparse array matrix code comprises:

computing ~~forming~~ a predetermined number of variables based on a virtual unrolling factor;

initializing the computed ~~formed~~ predetermined number of variables;

loading the prior computed values into the predetermined number of variables;

assigning the prior computed values to a predetermined number of substantially adjacent registers; and

software-pipelining using the assigned prior computed values.

17. (currently amended) The method of claim 16, further comprising:

performing register rotation to include computed values obtained from the current iteration; and

repeating the software-pipelining and using the register rotated computed values to reduce the recurrence initiation interval in a next iteration.

18. (original) The method of claim 17, wherein the computed values obtained from the predetermined number of prior adjacent iterations are based on virtual unrolling by using a virtual unrolling factor.

19. (original) The method of claim 18, wherein the virtual unrolling factor is three.

20. (original) A method comprising:

transforming loop computations from a predetermined number of prior adjacent iterations in sparse arrays/matrices to current loads code to perform a run time dependency check using a predetermined number of prior computed values;

software-pipelining the transformed loop computations from the predetermined number of prior adjacent iterations to perform a run time dependency check in a current iteration using the predetermined number of prior computed values; and

parallelizing the loop computations using the prior computed values.

21. (original) The method of claim 20, wherein the predetermined number of prior computed values is obtained from the predetermined number of prior adjacent iterations based on a virtual unrolling factor.

22. (original) The method of claim 21, wherein the virtual unrolling factor is three.

23. (original) The method of claim 20, wherein parallelizing the computations using the prior computed values comprise:

overlapping execution of dependence loop instructions in multiple dependent iterations in the sparse arrays/matrices using the software-pipelining.

24. (original) The method of claim 20, further comprising:

assigning the prior computed values to a predetermined number of adjacent register.

25. (original) The method of claim 24, further comprising:

performing register rotation to include computed values obtained from the current iteration.

26. (currently amended) An article comprising a computer-readable medium which stores computer-executable instructions, the instructions causing a computer to:

performing a run time dependency check during a current iteration using prior computed values obtained from a predetermined number of previous adjacent iterations in a sparse array matrix; and

parallelizing dependent ~~dependence~~ loop instructions between the current and subsequent multiple iterations by using the computed values to make the dependence loop instructions into independent loop iterations.

27. (currently amended) The article comprising a computer-readable medium which stores the computer executable instruction of claim 26, wherein performing the run time dependency check comprises:

transforming loop computations from a predetermined number of prior adjacent iterations in sparse arrays/matrices to current loads code to perform a run time dependency check using a predetermined number of prior computed values; and

software-pipelining the transformed loop computations from the predetermined number of prior adjacent iterations to perform a run time dependency check in a current iteration using the predetermined number of prior computed values.

28. (currently amended) The article comprising a computer-readable medium which stores the computer-executable instructions of claim 26, wherein the instructions further cause the [[a]] computer to assign the prior computed values to a predetermined number of adjacent register.

29. (currently amended) The article comprising a computer-readable medium which stores the computer-executable instructions of claim 28, wherein the instructions further cause the [[a]] computer to perform register rotation to include computed values obtained from the current iteration.

30. (currently amended) A system comprising:

a bus;

a processor coupled to the bus;

a memory coupled to the processor;

a network interface device;

wherein execution of loop computations in indirectly accessed sparse arrays/matrices ~~use~~ by the processor uses software-pipelining to increase instruction-level parallelism and decrease initiation interval by performing:

transforming loop computations from a predetermined number of prior adjacent iterations in sparse arrays/matrices to current loads code to perform a run time dependency check using a predetermined number of prior computed values;

software-pipelining the transformed loop computations from the predetermined number of prior adjacent iterations to perform a run time dependency check in a current iteration using the predetermined number of prior computed values; and

parallelizing the loop computations using the prior computed values to reduce recurrence initiation interval in the undisambiguated pointer stores from the predetermined number of prior adjacent iterations to current loads code based on the run time dependency check.

31. (original) The system of claim 30, wherein the processor further assigns the prior computed values to a predetermined number of adjacent register.

32. (original) The system of claim 31, wherein the processor further performs register rotation to include computed values obtained from the current iteration.

33. (original) A method comprising software-pipelining a loop by dynamic run time disambiguation of elements of an index array using otherwise-idle issues slots in a software-pipelining schedule.

34. (original) An article comprising a computer-readable medium that stores computer-executable instructions, the instructions causing a computer to software-pipeline a loop by dynamic run time disambiguation of elements of an index array using otherwise-idle issues slots in a software-pipelining schedule.

35. (new) The method of claim 33, wherein the dynamic run time disambiguation of the elements comprises:

holding a plurality of prior computed index variables associated with the index array in a corresponding plurality of rotating registers.

36. (new) The method of claim 33, wherein the dynamic run time disambiguation of the elements comprises:

comparing a currently-loaded value of an index variable associated with the index array to a prior computed value of the index variable.

37. (new) The article comprising a computer-readable medium which stores the computer-executable instructions of claim 34, wherein the instructions further cause the computer to use one of a plurality of prior computed index variables associated with the index array in place of a currently-loaded value of an index variable associated with the index array.

38. (new) The article comprising a computer-readable medium which stores the computer-executable instructions of claim 34, wherein the instructions further cause the computer to use a currently-loaded value of an index variable associated with the index array after determining that the currently-loaded value of the index variable is not equal to one of a plurality of prior computed index variables associated with the index array.